NVIDIA and Compositors

Andy Ritger, NVIDIA Corporation

October, 2014

Overview

- NVIDIA is working to support alternative window systems, such as Wayland and Mir.

- Goals:
  - NVIDIA driver to plug into window systems, in manner similar to Mesa-based drivers.
  - Leverage NVIDIA’s cross-platform OpenGL driver implementation.

Topics

In this talk, I'll try to address several topics:

- Our current areas of work to support Wayland/Mir:
  - KMS
  - X11-less EGL
  - Wayland client support in EGL (EGL_KHR_platform_wayland)
  - Wayland compositor support (EGL_WL_bind_wayland_display)
- Make a proposal for several EGL extensions to fill the role currently filled by GBM.

KMS

- In progress: refactoring our display programming support.
- Next: register with DRM with DRIVER_MODESET flag.
- Similar to how NVIDIA interfaces with DRM for Prime support.
- Goal is for NVIDIA driver to service DRM KMS ioctls.
  - xf86-video-modesetting, and other KMS clients should work on NVIDIA.
- NVIDIA's X driver won't use the KMS API directly
  - So NVIDIA X-based solutions continue to work on all existing platforms.
  - X driver will use the same refactored display code, so the same paths are exercised for X driver as any KMS clients.
- Hard part has been not regressing complex display features (e.g., G-Sync, FrameLock, SLI, Stereo, etc).

X11-less EGL

- NVIDIA's EGL and OpenGL implementation is used for both discrete GPUs and Tegra (starting with Tegra K1).
- X11-ful EGL support for discrete GPU: release 331.xx (autumn 2013).
- X11-less EGL: release 346.xx (autumn 2014)
  - But, KMS is not in 346.xx, so cannot yet display X11-less EGL.
- Besides KMS, also need mechanism for bootstrapping display + EGL.
  - More on this later in the talk...

EGL Support for Wayland Clients: EGL_KHR_platform_wayland

- Wayland clients need a way to create an EGLDisplay from a wl_display, and an EGLSurface from a wl_egl_surface.

- EGL_KHR_platform_wayland + EGL_EXT_platform_base define how to to this:

```
EGLDisplay dpy = eglGetPlatformDisplayEXT(EGL_PLATFORM_WAYLAND_EXT, wl_display, ...);
EGLSurface surf = eglCreatePlatformWindowSurfaceEXT(dpy, ..., wl_egl_surface, ...);
```

- Or, with "legacy" eglGetDisplay() path; e.g.,

```
EGLDisplay dpy = eglGetDisplay((EGLNativeDisplayType) wl_display);
```

EGL Support for Wayland Clients (continued)

- EGL implementation needs to:
  - Recognize the EGLDisplay and EGLSurface as Wayland-specific.
  - Coordinate with the instance of the EGL implementation loaded into the Wayland compositor; e.g., for buffer sharing between client and compositor.
- Make the EGL_KHR_platform_* support distinct from the core EGL implementation:
  - Let EGL_KHR_platform_ "plugins" register with the EGL implementation, so that they can be called for appropriate entry points.
  - Opportunity to standardize this platform plugin API?
  - Enable implementors of alternative platforms to implement the EGL platform support themselves.

EGL Support for Wayland Compositors: EGL_WL_bind_wayland_display

To share buffers between clients and compositors:

- Compositor uses EGL_WL_bind_wayland_display to bind a wl_display to an EGLDisplay.
  - EGL implementation registers a wayland extension, for use by the EGL implementation loaded into Wayland clients, to create wl_buffers.
- EGL_WL_bind_wayland_display also defines how to create an EGLImage from a wl_buffer.
  - Lets compositor texture from a client-created wl_buffer.

GBM

The remaining piece, currently used by compositors, is GBM.

This fills several roles for compositors:

- Bootstraps X11-less EGL: provides an EGL "platform" for the compositor.
- Mechanism for the compositor to import buffers from the client.
- Gives the compositor a way to name the buffers which should be submitted to KMS.

Based on the EGLDevice and EGLOutput ideas discussed at XDC 2013, we'd like to propose an alternative approach to filling the above roles.

X11-less EGL: Bootstrapping with GBM

- Bootstrap Mesa’s EGL through the use of GBM + DRM; e.g.,

```
int fd = drmOpen(...);
struct gbm_device *gbm_de = gbm_create_device(fd);
EGLDisplay dpy = eglGetDisplay(gbm_device);
```

I.e., use a gbm_device as the EGLNativeDisplayType argument to eglGetDisplay() .

X11-less EGL: Bootstrapping with EGLDevice

- EGL_EXT_device_base to enumerate the GPUs; e.g.,

```
EGLDeviceEXT egl_devices[32];
EGLint num_egl_devices;
eglQueryDevicesEXT(ARRAY_LEN(egl_devices),
                   egl_devices,
                   &num_egl_devices);
```

- Use eglQueryDeviceAttribEXT() to query EGLDevice properties.
  - The interesting EGLDevice properties defined by other extensions.
- Add additional extensions to query other EGLDevice properties.
  - E.g., PCI BusID, corresponding OpenCL or CUDA device ID, OpenWF WFD_DEVICE_ID, or DRM device file.

EGL_EXT_device_drm

- **EGL_EXT_device_drm defines EGL_DRM_DEVICE_FILE_EXT for use with eglQueryDeviceAttribEXT().**
- **Use this to correlate EGLDevices with DRM devices:**

```
char *drmDeviceFile =
    eglQueryDeviceStringEXT(egl_device, EGL_DRM_DEVICE_FILE_EXT);
```

- **EGL_EXT_platform_device defines EGL_PLATFORM_DEVICE_EXT for use with eglGetPlatformDisplay() to create an EGLDisplay on the EGLDevice. E.g.,**

```
EGLDisplay dpy =
    eglGetPlatformDisplay(EGL_PLATFORM_DEVICE_EXT, egl_device, ...);
```

EGL_EXT_output_base

- EGLDevice binds system native devices to EGL objects.
- EGLOutput does the same for native display-related objects.
- EGL_EXT_output_base:
  - Adds several new EGL objects:
    - EGLOutputLayerEXT: a surface; input to the display engine.
    - EGLOutputPortEXT: a "connector" in KMS terminology; output from the display engine.
  - Defines entry points to enumerate EGLOutputLayerEXTs and EGLOutputPortEXTs
  - Defines entry points to query/set properties on both new object types.
- EGL_EXT_output_* extensions define bindings to native objects.

EGL_EXT_output_drm

- EGL_EXT_output_drm maps DRM KMS to EGL objects:
  - Each KMS CRTC and each KMS plane maps to an EGLOutputLayerEXT.
  - Each KMS connector maps to an EGLOutputPortEXTs.

- The object mapping can be queried with eglQueryOutput{Layer,Port}{Attrib,String}EXT().

- The object mapping can be used when searching for EGLOutputLayerEXTs and EGLOutputPortEXTs.

EGL_EXT_output_drm sample usage

```
EGLOutputLayerEXT layer;
EGLint num_layers = 0;
const EGLAttrib layer_attribs[] = {
    EGL_DRM_PLANE_EXT, kms_plane_id,
    EGL_NONE,
};
eglGetOutputLayersEXT(dpy, layer_attribs, &layer, 1, &num_layers);

EGLOutputPortEXT port;
EGLint num_ports;
const EGLAttrib port_attribs[] = {
    EGL_DRM_CONNECTOR_EXT, kms_connector_id,
    EGL_NONE,
};
eglGetOutputPortsEXT(dpy, port_attribs, &port, 1, &num_ports);
```

KMS + EGL_EXT_device_drm + EGL_EXT_output_drm

- Using the EGL extensions described in the previous slides, compositors could use DRM's KMS API to set modes and correlate objects between KMS and EGL.
- The EGL extensions above are pretty trivial to implement.
- The interesting part is displaying content through these new EGL objects.
- Enter: EGLStreams

EGLStreams Background

- EGL_KHR_stream defines the EGLStream object.
  - A flexible mechanism for describing to EGL how to transfer frames between a "producer" and a "consumer".
- EGLStream cannot be used until consumer and producer are assigned.
- EGL_KHR_stream does not define consumers or producers itself; left to other extensions.
  - EGL_KHR_stream_producer_* extensions define how a producer produces a frame.
  - EGL_KHR_stream_consumer_* extensions define how a consumer consumes a frame.

EGLStreams Background (continued)

- By default, EGLStreams operate like a "one entry mailbox":
  - Producer conceptually replaces the mailbox content.
  - Consumer receives latest frame.
- EGL_KHR_stream_fifo: let EGLStreams operate as a FIFO.
- EGL_KHR_stream_cross_process_fd lets the EGLStream producer and consumer exist in different processes.
  - Process A: Create an EGLStream.
  - Process A: Get a file descriptor representing the EGLStream.
  - Use a UNIX domain socket to transfer the file descriptor from process A to process B.
  - Process B: Create an EGLStream from the file descriptor.

EGLStreams Background (continued)

- In-development extensions to make EGLStreams more flexible.
  - Be able to bind multiple consumers to an EGLStream.
  - Dynamically toggle the consumer of the EGLStream.
  - Dynamically resize (width, height) an EGLStream.

EGL_KHR_stream_producer_eglsurface

- **Example producer: EGL_KHR_stream_producer_eglsurface**

- **Create an EGLSurface as an EGLStream producer:**

```
EGLSurface surface =
    eglCreateStreamProducerSurfaceKHR(dpy, config, stream, attribs);
```

- **eglSwapBuffers() posts the frame in the EGLSurface to the EGLStream.**

EGL_KHR_stream_consumer_gltexture

- Example consumer: EGL_KHR_stream_consumer_gltexture
- Associate an OpenGL texture with an EGLStream using eglStreamConsumerGLTextureExternalKHR().
- Lock the current frame in the EGLStream for use as a texture using eglStreamConsumer{Acquire,Release}KHR().

EGLStreams: Desirable Properties

EGLStreams have some desirable properties:

- Explicit producers and consumers.
- Explicit transition points between producer's production and consumer's consumption.
- Encapsulation.

EGLStreams: Explicit Producers, Consumers

Why are explicit producers and consumers good?

- Driver can select optimal memory format and auxiliary resources that best suit the needs of the stated producers/consumers.
  - Otherwise, driver may have to assume the least common denominator of all possible producers and consumers.
  - In theory, possible to dynamically reformat based on current usage. But, this would be complex and error-prone.

EGLStreams: Explicit Transition Points

Why are explicit transition points good?

- When surface handoff is known, driver can resolve any synchronization or coherency requirements.
- Example: NVIDIA GPUs use color compression to reduce memory bandwidth usage (particularly important on Tegra)
  - 3D engine understands color compression, display does not.
  - Need to decompress, in-band, when handing off to display.
  - Decompression is expensive, so only do it when necessary.
- If driver knows producer/consumer + transition point:
  - Only do minimum sync/coherency resolution.
  - E.g., don’t need to decompress if consumer is texture, rather than display.

EGLStreams: Encapsulation

Why is encapsulation good?

- Encapsulation is a balancing act of providing an API that is:
  - Low-level enough to give clients the control they need.
  - High-level enough to let implementations make hardware-specific decisions, and not place undue burden and complexity upon API clients.
- Example: NVIDIA downsample-on-scanout:
  - When performing multisampled rendering, someone has to downsample.
  - Display engine can perform the downsampling during scanout.
  - If presentation from rendering through display is encapsulated within an API, then the driver implementation has the flexibility to take advantage of downsample-on-scanout when possible.

EGL_EXT_stream_consumer_egloutput

- EGLStream producer/consumer semantics match relationship of rendering and display engines on a GPU.
- EGL_EXT_stream_consumer_egloutput defines a way to make an EGLOutputLayerEXT the consumer of an EGLStream.
- We see this as the key to bootstrapping display of X11-less EGL.

Pseudocode

```
/* query the EGLDevices in the system */
EGLDeviceEXT egl_device;
EGLint num_egl_devices;
eglQueryDevicesEXT(1, &egl_device, &num_egl_devices);

/* get the device file name of the first EGLDevice */
char *drm_device_file = eglQueryDeviceStringEXT(egl_device, EGL_DRM_DEVICE_FILE_EXT);

/* open the DRM device file */
int drm_fd = open(drm_device_file);

/* Use DRM KMS to enumerate crtcs */
drmModeGetResources(drm_fd);
kms_crtc_id = ...
kms_plane_id = ...

/* set a mode on a crtc */
drmModeSetCrtc(drm_fd, kms_crtc_id, ...);

/* create an EGLDisplay on the EGLDevice */
EGLDisplay egl_dpy = eglGetPlatformDisplayEXT(EGL_PLATFORM_DEVICE_EXT, egl_device);

/* initialize EGL
...
```

Pseudocode (continued)

```
/* find the EGLOutputLayer that corresponds to the KMS plane */
EGLOutputLayerEXT egl_layer;
EGLint num_egl_layers;
EGLAttrib attrib_list[] = { EGL_DRM_PLANE_EXT, kms_plane_id, EGL_NONE };
eglGetOutputLayersEXT(egl_dpy, attrib_list, &egl_layer, 1, &num_egl_layers);

/* create a stream */
EGLStreamKHR egl_stream = eglCreateStreamKHR(egl_dpy, ...);

/* set the EGLOutputLayer as the consumer of the stream */
eglStreamConsumerOutputEXT(egl_dpy, egl_stream, egl_layer);

/* create an EGLSurface as the producer of the stream */
EGLSurface egl_surface = eglCreateStreamProducerSurfaceKHR(egl_dpy, ..., egl_stream, ...);

/* render stuff using OpenGL */
...

/* present to the stream: the content produced by the stream producer */
/* (egl_surface) is presented to the stream consumer (egl_layer) */
eglSwapBuffers(egl_dpy, egl_surface);
```

EGLStreams: Client/Compositor buffer sharing

Extend the EGL_WL_bind_wayland_display mechanism:

- **Add new eglQueryWaylandBufferWL() token: EGL_WAYLAND_BUFFER_TYPE_WL**
    - **Lets the compositor query the EGL "type" of the wl_buffer.**
- **Possible type is EGL_WAYLAND_BUFFER_EGLIMAGE_WL.**
- **Define new extension EGL_WL_wayland_buffer_eglstream**
    - **Adds new type: EGL_WAYLAND_BUFFER_EGLSTREAM_WL.**
    - **If wl_buffer type is EGLSTREAM_WL, then query fd of cross-process EGLStream:**

      ```
      eglQueryWaylandBufferWL(EGL_WAYLAND_BUFFER_EGLSTREAM_FD_WL)
      ```
- **The EGL implementation within the client could choose to make the wl_buffer's EGLSurface the stream producer.**
    - **Does not require changing Wayland clients.**

Why not GBM?

- GBM isn't bad; NVIDIA could work with it.

- Currently, libgbm is distributed as part of Mesa.
  - NVIDIA shouldn’t provide its own libgbm: libGL.so all over again.
  - To be fair: libgbm has a loadable backend, which could be extended to support loading vendor-specific GBM backends.

- However, we think the ecosystem can do better:
  - EGLStreams is an open standard.
  - EGLStreams is good for performance:
    - Defines clear producers and consumers, and clear transition points: lets driver implementations choose optimal resources, surface formats, synchronization, etc.
  - EGLStreams is portable. E.g., OpenWF Display + EGL + EGL_EXT_output_openwf on a platform without DRM, such as QNX.

EGL_EXT_stream_consumer_egloutput: Not Complete, Yet

- Interaction with KMS nuclear page flip
  - Cannot currently express atomicity for presentation across multiple EGLOutputLayerEXTs.
  - Could define additional EGLStreams extensions; e.g., bind several EGLOutputLayerEXTs together for purposes of atomic presentation.
  - KMS nuclear page flip would presumably be used by the Mesa implementation of an EGLOutputLayerEXT atomic presentation extension.
- Work through how to use EGLOutputLayerEXTs for clean transitions between console and compositors.
- Work out how EGLOutputLayerEXTs should be positioned within the KMS CRTC.

What is Next? For NVIDIA

For NVIDIA's part, we're going to continue to work on:

- KMS and registering as a KMS driver with DRM.
- Shipping EGL_EXT_device_base and EGL_EXT_platform_device
  - This is sufficient to create EGL+OpenGLES contexts without X11.
  - Shipping in our release 346.xx series, later this autumn.
  - Shipping EGL_KHR_platform_wayland, EGL_WL_bind_wayland_display, and EGL_WL_wayland_buffer_eglstream.
- Cleanup and post our Weston patches to demonstrate usage of EGLDevice + EGLOutput + EGLStreams.

What is Next? For other EGL implementers

For other EGL implementers:

- Consider the EGL extensions described in this talk.
  - The EGLDevice family of extensions are pretty simple to implement.
  - EGLStreams are less simple to implement.
- Provide feedback. We're interested in whether the community sees the EGLDevice + EGLOutput + EGLStreams proposal as a reasonable direction.

Thank You

Thanks to many who wrote and/or provided feedback on the EGL extensions described in this talk. In particular:

- James Jones (NVIDIA)
- Daniel Kartch (NVIDIA)
- Chad Versace (Intel)
- Acorn Pooley (formerly NVIDIA, now building robots somewhere)
- Christopher James Halse Rogers (Canonical)

and many others.

EGL extensions referenced in this talk:

http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_device_base.txt
http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_device_drm.txt
http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_platform_device.txt
http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_output_base.txt

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream.txt
http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream_fifo.txt
http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream_cross_process_fd.txt
http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream_producer_eglsurface.txt
http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream_consumer_gltexture.txt
http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_stream_consumer_egloutput.txt

http://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_platform_wayland.txt
http://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_platform_base.txt

http://cgit.freedesktop.org/mesa/mesa/tree/docs/specs/WL_bind_wayland_display.spec

https://github.com/aritger/xdc2014/blob/master/WL_bind_wayland_display.spec
https://github.com/aritger/xdc2014/blob/master/WL_wayland_buffer_eglstream.spec